Deterministic Single Exponential Time Algorithms for Connectivity Problems Parameterized by Treewidth*

Hans L. Bodlaender[1], Marek Cygan[2], Stefan Kratsch[3], and Jesper Nederlof[1]

[1] Utrecht University, The Netherlands
{H.L.Bodlaender,J.Nederlof}@uu.nl
[2] University of Warsaw, Poland
cygan@mimuw.edu.pl
[3] Technical University Berlin, Germany
stefan.kratsch@tu-berlin.de

Abstract. It is well known that many local graph problems, like Vertex Cover and Dominating Set, can be solved in $2^{\mathcal{O}(\mathtt{tw})}n^{\mathcal{O}(1)}$ time for graphs with a given tree decomposition of width $\mathtt{tw}$. However, for nonlocal problems, like the fundamental class of connectivity problems, for a long time it was unknown how to do this faster than $\mathtt{tw}^{\mathcal{O}(\mathtt{tw})}n^{\mathcal{O}(1)}$ until recently, when Cygan et al. (FOCS 2011) introduced the Cut&Count technique that gives randomized algorithms for a wide range of connectivity problems running in time $c^{\mathtt{tw}}n^{\mathcal{O}(1)}$ for a small constant c.

In this paper, we show that we can improve upon the Cut&Count technique in multiple ways, with two new techniques. The first technique (*rank-based approach*) gives deterministic algorithms with $O(c^{\mathtt{tw}}n)$ running time for connectivity problems (like Hamiltonian Cycle and Stei-ner Tree) *and* for *weighted* variants of these; the second technique (*determinant approach*) gives deterministic algorithms running in time $c^{\mathtt{tw}}n^{\mathcal{O}(1)}$ for *counting* versions, e.g., counting the number of Hamiltonian cycles for graphs of treewidth $\mathtt{tw}$.

The rank-based approach introduces a new technique to speed up dynamic programming algorithms which is likely to have more applications. The determinant-based approach uses the Matrix Tree Theorem for deriving closed formulas for counting versions of connectivity problems; we show how to evaluate those formulas via dynamic programming.

1 Introduction

It is known since the 1980s that many ($\mathcal{NP}$-hard) problems allow algorithms with a running time of the type $f(\mathtt{tw})n^c$ on graphs with n vertices and a tree decomposition of width $\mathtt{tw}$. It is a natural question how we can optimize on

* The second author is partially supported by NCN grant DEC-2012/05/D/ST6/03214 and Foundation for Polish Science. Part of the work of the third author was done at Utrecht University supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), project: 'KERNELS'. The fourth author is supported by the NWO project 'Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms'.

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 196–207, 2013.

the dependency of the treewidth, i.e., we first aim at obtaining a small growing (but, since we assume $\mathcal{P} \neq \mathcal{NP}$, exponential) function $f(\mathtt{tw})$ and second a small exponent c. For problems with locally checkable certificates, that is, certificates assigning a constant number of bits per node that can be checked by a cardinality check and iteratively looking at all neighborhoods of the input graph[1], it quickly became clear that $f(\mathtt{tw})$ only needs to be single exponential. See [14,20] for sample applications to the Independent Set/Vertex Cover problems. From the work of [13] and known Karp-reductions between problems it follows that this dependence cannot be improved to subexponential algorithms unless the Exponential Time Hypothesis (ETH) fails. In [17] it was shown that under the Strong ETH (SETH) the current algorithms are optimal even with respect to the bases of the exponential dependence on the treewidth, that is, problems with current best running time $c^{\mathtt{tw}}n^{\mathcal{O}(1)}$ cannot be solved in $(c-\epsilon)^{\mathtt{tw}}n^{\mathcal{O}(1)}$ for positive ϵ where, e.g., $c = 2$ for Independent Set and $c = 3$ for Dominating Set.

A natural class of problems that does not have locally checkable certificates are connectivity problems such as Hamiltonian Cycle and Steiner Tree (see for example [11, Section 5]), begging the question whether these can be solved within single exponential dependence on $\mathtt{tw}$ as well. Early results on the special case of H-minor-free graphs were given in [9]. A positive answer to the question was found by Cygan et al. [8] using a randomized approach termed "Cut & Count": It provided a transformation of the natural certificates to "cut-certificates" transforming the connectivity requirement into a locally checkable requirement. The transformation is only valid modulo 2, but by a standard technique introducing randomization [19], the decision variant can be reduced to the counting modulo 2 variant. This result was considered surprising since in the folklore $2^{\mathcal{O}(\mathtt{tw}\log\mathtt{tw})}n^{\mathcal{O}(1)}$ dynamic programming routines for connectivity problems all information stored seemed needed: Given two partial solutions inducing different connectivity properties, one can be extendable to a solution while the other one can not (this resembles the notion of Myhill-Nerode equivalence classes [12]).

The Cut & Count approach is one of the dynamic programming algorithms using a modulo 2 based transformation [3,4,16,15,18,23]. These algorithms give the smallest running times currently known, but have several disadvantages compared to traditional dynamic programming algorithms: (a) They are randomized. (b) The dependence on the inputs weights in weighted extensions is pseudo-polynomial. (c) They do not extend to counting the number of witnesses. (d) They do not give intuition for the optimal substructure / equivalence classes. An additional disadvantage of the Cut & Count approach of [8], compared to traditional dynamic programming algorithms on tree decompositions, is that their dependence in terms of the input graph is superlinear. Our work shows that each of these disadvantages can be overcome, with two different approaches, both giving deterministic algorithms for connectivity problems that are single exponential in the treewidth.

[1] E.g., for the odd cycle transversal problem that asks to make the input graph bipartite by removing at most k vertices, a locally checkable certificate would be a solution set combined with a proper two-coloring of the remaining graph.

Table 1. Our results for some famous computational problems. The second column gives running times when a path decomposition of width $\mathtt{pw}$ is given in the input; the third column gives running times when a tree decomposition of width $\mathtt{tw}$ is given in the input. Rows 1–4 use the rank-based approach; Rows 5–7 the determinant approach. ω denotes the matrix multiplication exponent (currently it is known that $\omega < 2.3727$ [24]).

1	Weighted Steiner Tree	$n(1+2^{\omega})^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$	$n(1+2^{\omega+1})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$
2	Traveling Salesman	$n(2+2^{\omega/2})^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$	$n(5+2^{(\omega+2)/2})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$
3	k-Path	$n(2+2^{\omega/2})^{\mathtt{pw}}(k+\mathtt{pw})^{\mathcal{O}(1)}$	$n(5+2^{(\omega+2)/2})^{\mathtt{tw}}(k+\mathtt{tw})^{\mathcal{O}(1)}$
4	Feedback Vertex Set	$n(1+2^{\omega})^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$	$n(1+2^{\omega+1})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$
5	# Hamiltonian Cycle	$\tilde{\mathcal{O}}(6^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^2)$	$\tilde{\mathcal{O}}(15^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^2)$
6	# Steiner Tree	$\tilde{\mathcal{O}}(5^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^3)$	$\tilde{\mathcal{O}}(10^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^3)$
7	Feedback Vertex Set	$\tilde{\mathcal{O}}(5^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^3)$	$\tilde{\mathcal{O}}(10^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^3)$

Our Contribution. We present the "Rank based" and "Squared determinant" approaches. For a number of key problems, we state the results obtained by applying these approaches in Table 1.

The approaches can be used to quickly and deterministically solve weighted and counting versions of problems solved by the Cut & Count approach. Additional advantages of the rank based approach are that it gives a more intuitive insight in the optimal substructure / equivalence classes of a problem and that is has only a linear dependence on the input graph in the running time. The only disadvantage of both approaches when compared to the Cut & Count approach is that the dependence on the treewidth or pathwidth in the running time is slightly worse. However, although we did not manage to overcome it, this disadvantage might not be inherently due to the new methods. Due to the generality of our key ideas, as one might expect, the approaches can be applied to other connectivity problems, such as all problems mentioned in [8]. However, our methods may inspire future work not involving tree decompositions as well.

Since one of the main strengths of the treewidth concept seems to be its ubiquity, it is perhaps not surprising that our results improve, simplify, generalize or unify a number of seemingly unrelated results. E.g., we unify algorithms for Feedback Vertex Set [6] and k-Path [2] and generalize algorithms for restricted inputs such as H-minor-free (e.g. [9]) or bounded degree graphs (e.g. [10]). A more detailed discussion is postponed to the full version.

For space reasons, many details (and material that might be considered 'more than details') are omitted from this extended abstract. For all such material, we refer to the full paper, available at arXiv.org [5].

2 Preliminaries

Partitions and the Partition Lattice. Given a base set U, we use $\Pi(U)$ for the set of all partitions of U. It is known that, together with the coarsening relation $\sqsubseteq$, $\Pi(U)$ gives a lattice, with the minimum element being $\{U\}$ and the maximum element being the partition into singletons. We denote $\sqcap$ for the

meet operation and $\sqcup$ for the join operation in this lattice; these operators are associative and commutative. We use $\Pi_2(U) \subset \Pi(U)$ to denote the set of all partitions of U in blocks of size 2, or equivalently, the set of perfect matchings over U. Given $p \in \Pi(U)$, we let $\#\texttt{blocks}(p)$ denote the number of blocks of p. If $X \subseteq U$ we let $p_{\downarrow X} \in \Pi(X)$ be the partition obtained by removing all elements not in X from it, and analogously we let for $U \subseteq X$ denote $p_{\uparrow X} \in \Pi(X)$ for the partition obtained by adding singletons for every element in $X \setminus U$ to p. Also, for $X \subseteq U$, we let $U[X]$ be the partition of U where one block is $\{X\}$ and all other blocks are singletons. If $a, b \in U$ we shorthand $U[ab] = U[\{a, b\}]$. The empty set, vector and partition are all denoted by $\emptyset$.

Tree Decompositions and Treewidth. A *tree decomposition* [22] of a graph G is a tree $\mathbb{T}$ in which each node x has an assigned set of vertices $B_x \subseteq V$ (called a *bag*) such that $\bigcup_{x \in \mathbb{T}} B_x = V$ with the following properties: (i) for any $uv \in E$, there exists an $x \in \mathbb{T}$ such that $u, v \in B_x$, (ii) if $v \in B_x$ and $v \in B_y$, then $v \in B_z$ for all z on the (unique) path from x to y in $\mathbb{T}$. In a *nice tree decomposition*, $\mathbb{T}$ is rooted, and each bag is of one of the following types:

- **Leaf Bag:** a leaf x of $\mathbb{T}$ with $B_x = \emptyset$.
- **Introduce Vertex Bag:** an internal vertex x of $\mathbb{T}$ with one child vertex y for which $B_x = B_y \cup \{v\}$ for some $v \notin B_y$.
- **Introduce Edge Bag:** an internal vertex x of $\mathbb{T}$ labeled with an edge $uv \in E$ with one child bag y for which $u, v \in B_x = B_y$.
- **Forget Bag:** an internal vertex x of $\mathbb{T}$ with one child bag y for which $B_x = B_y \setminus \{v\}$ for some $v \in B_y$.
- **Join Bag:** an internal vertex x with two child vertices l and r with $B_x = B_r = B_l$.

We require that every edge in E is introduced exactly once. This variant of nice tree decompositions was also used by Cygan et al. [8].

A nice tree decomposition is a *nice path decomposition* if it does not contain join bags. The *width* $tw(\mathbb{T})$ of a (nice) tree decomposition $\mathbb{T}$ is the size of the largest bag of $\mathbb{T}$ minus one, and the treewidth (pathwidth) of a graph G can be defined as the minimum treewidth over all nice tree decompositions (nice path decompositions) of G.

In this paper, we will always assume that nice tree decompositions of the appropriate width are given. To each bag x in a nice tree decomposition $\mathbb{T}$, we associate the graph $G_x = (V_x, E_x)$ with V_x the union of all bags B_y with y a descendant of x, and E_x the set of all edges introduced in an descendant of x.

Further Notation. For two integers a, b we use $a \equiv b$ to indicate that a is even if and only if b is even. We use $\mathbb{N}$ to denote the set of all non-negative integers. We use Iverson's bracket notation: if p is a predicate we let $[p]$ be 1 if p is true and 0 otherwise. If $\omega : U \to \{1, \ldots, N\}$, we shorthand $\omega(S) = \sum_{e \in S} \omega(e)$ for $S \subseteq U$. For a function/vector s by $s[v \to \alpha]$ we denote the function $s \setminus \{(v, s(v))\} \cup \{(v, \alpha)\}$. Note that this definition works regardless of whether $s(v)$ is already defined or not. We use either $s|_X$ or $s_{|X}$ to denote the function obtained by restricting the domain to X.

3 The Rank Based Approach

3.1 Main Ideas of the Approach

Recall that a dynamic programming algorithm fixes a way to decompose certificates into 'partial certificates', and builds partial certificates in a bottom-up manner while storing only their essential information. Given some language $L \subseteq \{0,1\}^*$, this is typically implemented by defining an equivalence $\sim$ on partial certificates $x, y \in \{0,1\}^k$ such that $x \sim y$ if $xz \in L \leftrightarrow yz \in L$, for every extension $z \in \{0,1\}^l$. For connectivity problems on treewidth, the number of non-equivalent certificates can be seen to be $2^{\Theta(\mathtt{tw} \cdot \lg \mathtt{tw})}$. See for example [21] for a lower bound in communication complexity.

We will use however, that sometimes we can represent the joint essential information for sets of partial certificates more efficiently than naively representing essential information for every partial certificate separately. The rank based approach achieves this as follows: Given a dynamic programming algorithm, consider the matrix A whose rows and columns are indexed by partial certificates, with $A[x, y] = 1$ if and only if $xy \in L$. Then observe that if a set of rows $X \subseteq \{0,1\}^n$ is linearly dependent (modulo 2), any partial certificate $x \in X$ is redundant in the sense that if $xz \in L$, there will be $y \in X, y \neq x$ with $yz \in L$. Hence, the essential information can be reduced to $\mathrm{rk}(A)$ partial certificates.

The second ingredient to our approach are proofs that for the considered problems, the rank (working in GF(2)) of such a matrix A is single exponential in the treewidth, and moreover, we can give explicit bases.

Now, the approach is as follows: take the 'usual' dynamic programming algorithm for the problem at hand, but add the following step: after each computation of a table at a bag node, form the submatrix of A with for each entry in the table a row and for each element of the basis a column; find a row-basis of this matrix and continue with only the partial certificates in this basis, of which there are not more than the rank of the certificate matrix. This is easily extended to weighted problems using a minimum weighted row-basis. For getting this approach to work for connectivity problems we require an upper bound on the rank of the matrix $\mathcal{M}$ defined as follows: Fix a ground set U and let p and q be partitions of U (p and q represent connectivity induced by partial solutions), define $\mathcal{M}[p, q]$ to be 1 if and only if the meet of p and q is the trivial partition, that is, if the union of the partial solutions induce a connected solution. Although this matrix has dimensions of order $2^{\theta(|U| \log |U|)}$, we exploit a simple factorization in GF(2) of matrices with inner dimension $2^{|U|}$ using an idea of [8]. To avoid creating a series of ad hoc results for single problems, we introduce a collection of operations on sets of weighted partitions, such that our results apply to any dynamic programming (DP) formulation that can be expressed using these operators only (see Section 3.2). In this extended abstract, we only state and illustrate the main building blocks of the approach.

3.2 Operators on Sets of Weighted Partitions

Recall that $\Pi(U)$ denotes the set of all partitions of some set U. A *set of weighted partitions* is a set $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$, i.e., a family of pairs, each consisting of a partition of U and a non-negative integer weight.

We now define a collection of operators on sets of weighted partitions. The operators naturally apply to connectivity problems by allowing, e.g., gluing of connected components (i.e., different sets in a partition), or joining of two partial solutions by taking the meet operation $\sqcap$ on the respective partitions.

An important reason of interest in these operators is the following: if the recurrences in a dynamic programming algorithm on a tree decomposition only use these operators, then the naive algorithm evaluating the recurrence can be improved beyond the typical $2^{\Omega(\mathtt{tw}\cdot\log \mathtt{tw})}$ that comes from the high number of different possible partial solutions; and we typically get a running time of $O(c^{\mathtt{tw}}n)$.

For notational ease, we let $\mathtt{rmc}(\mathcal{A})$ denote the set obtained by removing non-minimal weight copies, i.e., $\mathtt{rmc}(\mathcal{A}) = \{(p,w) \in \mathcal{A} \mid \nexists (p,w') \in \mathcal{A} \wedge w' < w\}$.

Definition 1. *Let U be a set and $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$.*

- ***Union.*** *For $\mathcal{B} \subseteq \Pi(U) \times \mathbb{N}$, define $\mathcal{A} \uplus \mathcal{B} = \mathtt{rmc}(\mathcal{A} \cup \mathcal{B})$.*
- ***Insert.*** *For $X \cap U = \emptyset$, define $\mathtt{ins}(X, \mathcal{A}) = \{(p_{\uparrow U \cup X}, w) \mid (p,w) \in \mathcal{A}\}$.*
- ***Shift.*** *For $w' \in \mathbb{N}$ define $\mathtt{shft}(w', \mathcal{A}) = \{(p, w+w') \mid (p,w) \in \mathcal{A}\}$.*
- ***Glue.*** *For u, v, let $\hat{U} = U \cup \{u,v\}$ and define $\mathtt{glue}(uv, \mathcal{A}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as $\mathtt{glue}(uv, \mathcal{A}) = \mathtt{rmc}(\{(\hat{U}[uv] \sqcap p_{\uparrow \hat{U}}, w) \mid (p,w) \in \mathcal{A}\})$. Also, if $\omega : \hat{U} \times \hat{U} \to \mathbb{N}$, let $\mathtt{glue}_\omega(uv, \mathcal{A}) = \mathtt{shft}(\omega(u,v), \mathtt{glue}(uv, \mathcal{A}))$.*
- ***Project.*** *For $X \subseteq U$ let $\overline{X} = U \setminus X$, and define $\mathtt{proj}(X, \mathcal{A}) \subseteq \Pi(\overline{X}) \times \mathbb{N}$ as $\mathtt{proj}(X, \mathcal{A}) = \left\{ (p_{\downarrow \overline{X}}, w) \,\middle|\, (p,w) \in \mathcal{A} \wedge \forall e \in X : \exists e' \in \overline{X} : p \sqsubseteq U[ee'] \right\}$.*
- ***Join.*** *For $\mathcal{B} \subseteq \Pi(U') \times \mathbb{N}$ let $\hat{U} = U \cup U'$ and define $\mathtt{join}(\mathcal{A}, \mathcal{B}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as $\mathtt{join}(\mathcal{A}, \mathcal{B}) = \mathtt{rmc}(\left\{ (p_{\uparrow \hat{U}} \sqcap q_{\uparrow \hat{U}}, w_1 + w_2) \,\middle|\, (p,w_1) \in \mathcal{A} \wedge (q, w_2) \in \mathcal{B} \right\})$.*

See the full version for a description of the operators in words. Using straightforward implementation each of the operations union, shift, insert, glue and project can be performed in $S|U|^{\mathcal{O}(1)}$ time where S is the size of the input of the operation. Given $\mathcal{A}$ and $\mathcal{B}$, $\mathtt{join}(\mathcal{A}, \mathcal{B})$ can be computed in time $|\mathcal{A}| \cdot |\mathcal{B}| \cdot |U|^{\mathcal{O}(1)}$.

3.3 Representing Collections of Partitions

The key idea for getting a faster dynamic programming algorithm is to follow the naive DP, but to consider only small representative sets of weighted partitions instead of all weighted partitions that would be considered by the naive DP. Intuitively, a representative (sub)set of partial solutions should allow us to always extend to an optimal solution provided that one of the complete set of partial solutions extends to it. Let us define this formally.

Definition 2. *Given a set of weighted partitions $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$ and a partition $q \in \Pi(U)$, define $\mathtt{opt}(q, \mathcal{A}) = \min\{w \mid (p,w) \in \mathcal{A} \wedge p \sqcap q = \{U\}\}$. For another set of weighted partitions $\mathcal{A}' \subseteq \Pi(U) \times \mathbb{N}$, we say that $\mathcal{A}'$* represents *$\mathcal{A}$ if for all $q \in \Pi(U)$ it holds that $\mathtt{opt}(q, \mathcal{A}') = \mathtt{opt}(q, \mathcal{A})$.*

Note that the definition of representation is symmetric, i.e., if $\mathcal{A}'$ represents $\mathcal{A}$ then $\mathcal{A}$ also represents $\mathcal{A}'$. However, we will only be interested in the special case where $\mathcal{A}' \subseteq \mathcal{A}$ and where we have a size guarantee for finding a small such subset $\mathcal{A}'$.

Definition 3. *A function* $f : 2^{\Pi(U)\times\mathbb{N}} \times Z \to 2^{\Pi(U')\times\mathbb{N}}$ *is said to* preserve representation *if for every* $\mathcal{A}, \mathcal{A}' \subseteq \Pi(U) \times \mathbb{N}$ *and* $z \in Z$ *it holds that if* $\mathcal{A}'$ *represents* $\mathcal{A}$ *then* $f(\mathcal{A}', z)$ *represents* $f(\mathcal{A}, z)$*. (Note that* Z *stands for any combination of further inputs.)*

The following lemma and theorem are our main building blocks and establish that the operations needed for the DP preserve representation, and, crucially, that we can always find a reasonably small representative set of weighted partitions. The proofs are deferred to the full version.

Lemma 1. *The union, insert, shift, glue, project, and join operations from Definition 1 preserve representation.*

Theorem 1. *There is an algorithm* `reduce()` *that given set of weighted partitions* $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$ *takes time* $|\mathcal{A}|2^{(\omega-1)|U|}|U|^{\mathcal{O}(1)}$ *and outputs a set of weighted partitions* $\mathcal{A}' \subseteq \mathcal{A}$ *such that* $\mathcal{A}'$ *represents* $\mathcal{A}$ *and* $|\mathcal{A}'| \leq 2^{|U|}$*, where* ω *denotes the matrix multiplication exponent (it is known that* $\omega < 2.3727$ *[24]).*

With help of Lemma 1 and Theorem 1, we can now handle problems in the following way: we give recurrences for the different types of bags (leaf, introduce, etc.) in a nice tree (or path) decompositions that use only union, insert, shift, glue, project, and join operations. Correctness of an algorithm that interleaves the computation of the recurrences with the (Gaussian elimination based) reduction step of Theorem 1 follows. Inspection of the resulting algorithms is still needed for establishing the precise time bounds.

3.4 Application to Steiner Tree

We now sketch how to solve the Steiner Tree problem via a dynamic programming formulation that requires only the operators introduced in Section 3.4.

Steiner Tree
Input: A graph $G = (V, E)$ weight function $\omega\colon E \to \mathbb{N} \setminus \{0\}$, a terminal set $K \subseteq V$ and a nice tree decomposition $\mathbb{T}$ of G of width `tw`.
Question: The minimum of $\omega(X)$ over all subsets $X \subseteq E$ of G such that $G[X]$ is connected and $K \subseteq V(G[X])$.

We now describe the 'folklore' dynamic programming algorithm for Steiner Tree on nice tree decompositions. For each bag x, we compute a table A_x. A_x has an entry for each $\mathbf{s} \in \{0,1\}^{B_x}$; this entry is a set of pairs consisting of a partition of $s^{-1}(1)$ and a weight value. The intuition is as follows: a 'partial solution' for the Steiner tree problem is a forest F in G_x that contains all vertices in K and each tree in the forest has a nonempty intersection with B_x; now $\mathbf{s}$

denotes which vertices in B_x belong to F, i.e., $v \in B_x$ belongs to F, iff $s(v) = 1$; and the partition tells which vertices in $s^{-1}(1)$ belong to the same tree in F. The weight value gives the total weight of all edges in F; for given $\mathbf{s}$ and partition p, we only store the minimum weight over all applicable forests.

Formally, we define, for a bag x, and $\mathbf{s} \in \{0,1\}^{B_x}$

$$A_x(\mathbf{s}) = \left\{ \left(p, \min_{X \in \mathcal{E}_x(p,\mathbf{s})} \omega(X) \right) \middle| \, p \in \Pi(s^{-1}(1)) \wedge \mathcal{E}_x(p,\mathbf{s}) \neq \emptyset \right\}$$

with $\mathcal{E}_x(p,\mathbf{s}) = \{X \subseteq E_x |\ \forall v \in B_x : v \in V(G[X]) \vee v \in K \rightarrow s(v) = 1 \wedge \forall v_1, v_2 \in s^{-1}(1) : v_1 v_2$ are in same block in $p \leftrightarrow v_1, v_2$ connected in $G[X] \wedge \#\mathtt{blocks}(p) = \mathtt{cc}(G[X])\}$.

The table A_x can be computed from the tables A_y of all the children of y, using only the union, insert, shift, glue, project, and join operations. For each of the different types of bags (leaf, forget, etc.), we can give such a recurrence. We only state here the recurrence for the Introduce edge bag; for the other recurrences, we refer to the full version.

Consider a bag x with child y introducing edge $e = uv$. One can show the following recurrence, which is, by the results above, representation preserving.

$$A_x(\mathbf{s}) = \begin{cases} A_y(\mathbf{s}) & \text{if } s(u) = 0 \vee s(v) = 0, \\ A_y(\mathbf{s}) \uplus \mathtt{glue}_\omega(uv, A_y(\mathbf{s})), & \text{otherwise.} \end{cases}$$

Theorem 2. *There exist algorithms that solve* Steiner Tree *in time* $n(1 + 2^\omega)^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}$ *time if a path decomposition of width* $\mathtt{pw}$ *of* G *is given, and in time* $n(1 + 2^{\omega+1})^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}$ *time if a tree decomposition of width* $\mathtt{tw}$ *of* G *is given.*

Proof. The algorithm is the following: use the above dynamic programming formulation as discussed to compute A_r (where r is the child of the root, as discussed), but after evaluation of any entry A_x, use Theorem 1 to obtain and store $A'_x = \mathtt{reduce}(A_x)$ rather than A_x. Since $A_x = \mathtt{reduce}(\mathcal{A})$ represents $\mathcal{A}$ and the recurrence uses only the operators defined in Definition 1 which all preserve representation by Lemma 1, we have as invariant that for every $x \in \mathbb{T}$ the entry A'_x stored for A_x represents A_x by Lemma 1. In particular, the stored value $A'_r(\mathbf{s})$ represents $A_r(\mathbf{s})$ and hence we can safely read off the answer to the problem from this stored value as done from $A_r(\mathbf{s})$ in the folklore dynamic programming algorithm. The time analysis can be found in the full version. □

3.5 Further Results

A large number of other problems can be handled with the rank based approach. In Table 1 we give a number of key results. Details are in the full version. The results on Hamiltonian Cycle and Traveling Salesman are obtained by combining our approach with the following result.

Theorem 3 ([7]). *Let* $\mathcal{H}$ *be the submatrix of* $\mathcal{M}$ *restricted to all matchings. Then* $\mathcal{H}$ *can be factorized into two matrices whose entries can be computed in time* $|U|^{\mathcal{O}(1)}$*, where the inner dimension of the factorization is* $2^{|U|/2-1}$.

An interesting corollary of our work is the following.

Theorem 4. *There is an algorithm solving* Traveling Salesman *on cubic graphs in* $1.2186^n n^{\mathcal{O}(1)}$ *time.*

4 Determinant Approach

In this section we will present the determinant approach that can be used to solve counting versions of connectivity problems on graphs of small treewidth. Throughout this section, we will assume a graph G along with a path/tree decomposition $\mathbb{T}$ of G of width `pw` or `tw` is given.

4.1 Main Ideas of the Approach

The determinant approach gives a generic transformation of counting connected objects to a more local transformation. In [8] the existence of such a transformation in GF(2) was already given. For extending this to counting problems, we will need three key insights. The first insight is that (a variant of) Kirchhoff's Matrix Tree Theorem gives a reduction from counting connected objects to computing a sum of determinants. However, we cannot fully control the contribution of a connected object (it will appear to be either 1 or -1). To overcome this we ensure that every connected object contributed exactly once, we compute a sum of *squares of determinants*. The last obstacle is that the computation of a determinant is not entirely local (in the sense that we can verify a contribution by iteratively considering its intersection with all bags) since we have to account for the number of inversions of a permutation in every summand of the determinant. To overcome this obstacle, we show that this computation becomes a local computation once we have fixed the order of the vertices in a proper way.

Formally, let A be an incidence matrix of an orientation of G, that is $A = (a_{i,j})$ is a matrix with n rows and m columns. Each row of A is indexed with a vertex and each column of A is indexed with an edge. The entry $a_{v,e}$ is defined to be 0 if $v \notin e$; -1 if $e = uv$ and $u < v$; or 1 if $e = uv$ and $u > v$. We assume, that all the vertices are ordered with respect to the post-ordering of their forget nodes in the given tree decomposition, that is vertices forgotten in a left subtree are smaller than vertices forgotten in the right subtree, and a vertex forgotten in a bag x is smaller than a vertex forgotten in a bag which is an ancestor of x. Similarly we order edges according to the post-ordering of the introduce edge nodes in the tree decomposition.

Let v_1 be an arbitrary fixed vertex and let F be the matrix A with the row corresponding to v_1 removed. For a subset $S \subseteq E$ let F_S be the matrix with $n-1$ rows and $|S|$ columns, whose columns are those of F with indices in S. The following folklore lemma is used in the proof of the Matrix Tree Theorem (see for example [1, Page 203] where our matrix is denoted by N).

Lemma 2. *Let $S \subseteq E$ be a subset of size $n-1$. If (V, S) is a tree, then $|\det(F_S)| = 1$ and $\det(F_S) = 0$ otherwise.*

We are up to compute the number of connected edgesets X such that $X \in \mathcal{F}$ where $\mathcal{F}$ is some implicitly defined set family. Our main idea is to use Lemma 2 to reduce this task to computing the quantity $\sum_{X \in \mathcal{F}} \det(F_X)^2$ instead, and to ensure that if $X \in \mathcal{F}$ is connected, then it is a tree.

For two (not necessarily disjoint) subsets V_1, V_2 of an ordered set let us define $\mathrm{inv}(V_1, V_2) = |\{(u, v) : u \in V_1, v \in V_2, u > v\}|$. If X, Y are ordered sets, recall that for a permutation $f : X \overset{1-1}{\to} Y$ we have that the sign equals $\mathrm{sgn}(f) = (-1)^{|\{(e_1,e_2):e_1,e_2 \in S \wedge e_1 < e_2 \wedge f(e_1) > f(e_2)\}|}$. The following proposition will be useful:

Proposition 1. *Let $X_l, X_r \subseteq V$ and $Y_l, Y_r \subseteq E$ such that $X_l \cap X_r = \emptyset$ and $Y_l \cap Y_r = \emptyset$, and for every $e_1 \in Y_l$ and $e_2 \in Y_r$ we have that $e_1 < e_2$. Suppose $f_l : Y_l \overset{1-1}{\to} X_l$ and $f_r : Y_r \overset{1-1}{\to} X_r$. Denote $f = f_l \cup f_r$, that is, $f(v) = f_l(v)$ if $v \in Y_l$ and $f(v) = f_r(v)$ if $v \in Y_r$. Then it holds that $\mathrm{sgn}(f) = \mathrm{sgn}(f_1)\mathrm{sgn}(f_2) \cdot (-1)^{\mathrm{inv}(X_l, X_r)}$.*

To see that the proposition is true, note that from the definition of sgn, the pairs e_1, e_2 with $e_1, e_2 \in Y_1$ or $e_1, e_2 \in Y_2$ are already accounted for in the part $\mathrm{sgn}(f_1)\mathrm{sgn}(f_2)$ so it remains to show that $|\{(e_1, e_2) : e_1 \in Y_1, e_2 \in Y_2 \wedge e_1 < e_2 \wedge f(e_1) > f(e_2)\}|$ indeed equals $\mathrm{inv}(X_l, X_r)$, which follows easily from the assumption that $e_1 < e_2$.

4.2 Counting Hamiltonian Cycles

For our first application to counting Hamiltonian cycles, we derive the following formula which expresses the number of Hamiltonian cycles of a graph. (We use that a 2-regular graph has n subtrees on $n-1$ edges if it is connected and 0 otherwise, and Lemma 2.)

$$\begin{aligned}
&\sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} [X \text{ is a Hamiltonian cycle}] \\
&= \frac{1}{n} \cdot \sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} \sum_{S \subseteq X, |S| = n-1} [(V, S) \text{ is a tree}] \\
&= \frac{1}{n} \cdot \sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} \sum_{S \subseteq X, |S| = n-1} \det(F_S)^2.
\end{aligned}$$

By plugging in the permutation definition of a determinant, we obtain the following expression for the number of Hamiltonian cycles of a graph:

$$\begin{aligned}
&\frac{1}{n} \sum_{X \subseteq E; \forall_{v \in V} \deg_X(v) = 2} \sum_{S \subseteq X; |S| = n-1} \Big(\sum_{f : S \overset{1-1}{\to} V \setminus \{v_1\}} \mathrm{sgn}(f) \prod_{e \in S} a_{f(e), e} \Big)^2 \\
=&\frac{1}{n} \sum_{\substack{X \subseteq E \\ \forall_{v \in V} \deg_X(v) = 2}} \sum_{S \subseteq X} \sum_{f_1, f_2 : S \overset{1-1}{\to} V \setminus \{v_1\}} \mathrm{sgn}(f_1)\mathrm{sgn}(f_2) \prod_{e \in S} a_{f_1(e), e} a_{f_2(e), e}.
\end{aligned}$$

Note that in the last equality we dropped the assumption $|S| = n - 1$, as it follows from the fact that f_1 (and f_2) is a bijection.

Our goal is to compute the formula by dynamic programming over some nice tree decomposition $\mathbb{T}$. To this end, let us define a notion of "partial sum" of the above formula, that we will store in our dynamic programming table entries. For every bag $x \in \mathbb{T}$, $s_{\deg} \in \{0,1,2\}^{B_x}$, $s_1 \in \{0,1\}^{B_x}$ and $s_2 \in \{0,1\}^{B_x}$ define $A_x(s_{\deg}, s_1, s_2) =$

$$\sum_{\substack{X \subseteq E_x \\ \forall_{v \in (V_x \setminus B_x)} \deg_X(v)=2 \\ \forall_{v \in B_x} \deg_X(v)=s_{\deg}(v)}} \sum_{S \subseteq X} \sum_{\substack{f_1 : S \overset{1-1}{\to} (V_x \setminus \{v_1\}) \setminus s_1^{-1}(0) \\ f_2 : S \overset{1-1}{\to} (V_x \setminus \{v_1\}) \setminus s_2^{-1}(0)}} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e \in S} a_{f_1(e),e} a_{f_2(e),e}.$$

Intuitively in $s_{\deg}$ we store the degrees of vertices of B_x in $G[X]$, whereas s_1 (and s_2) specify whether a vertex of B_x was already used as a value of the bijection f_1 (and f_2).

Showing that the terms $A_x(s_{\deg}, s_1, s_2)$ can be computed in the claimed time for the different types of nodes in a nice tree decompositions requires an intricate proof, for which we refer to the full version.

Theorem 5. *There exist algorithms that given a graph G solve* #Hamiltonian Cycle *in $\tilde{\mathcal{O}}(6^{\mathtt{pw}}\mathtt{pw}^{\mathcal{O}(1)}n^2)$ time if a path decomposition of width* `pw` *is given, and in time $\tilde{\mathcal{O}}(15^{\mathtt{tw}}\mathtt{tw}^{\mathcal{O}(1)}n^2)$ time if a tree decomposition of width* `tw` *is given.*

For other results that can be obtained with this approach, see the full paper.

5 Conclusions

In this paper, we have given deterministic algorithms for connectivity problems on graphs of small treewidth, with the running time only single exponential in the treewidth. We have given two different techniques. Each technique solves the standard versions, but for the counting and weighted variants, only one of the techniques appears to be usable. The rank-based approach gives a new twist to the dynamic programming approach, in the sense that we consider the "algebraic structure" of the partial certificates in a quite novel way. This suggests a study of this algebraic structure for dynamic programming algorithms for other problems.

In related work [7], an approach similar to the rank-based one, but focused on perfect matchings instead of partitions, is used to obtain a faster randomized algorithm for Hamiltonicity parameterized by pathwidth; the algorithm is showed to be tight under SETH, but does not apply to counting or the weighted case. The present results, while slower for the special case of Hamiltonicity, give deterministic algorithms that apply also to problems where connectivity of partial solutions does not appear to allow encoding via perfect matchings (e.g., Steiner Tree and Feedback Vertex Set), and to counting and weighted versions.

Acknowledgements. We thank Piotr Sankowski for pointing us to relevant literature on matrix-multiplication algorithms and Marcin Pilipczuk and Łukasz Kowalik for helpful discussions at an early stage of the paper.

References

1. Aigner, M., Ziegler, G.: Proofs from the book, Berlin, Germany (2010)
2. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM 42(4), 844–856 (1995)
3. Björklund, A.: Determinant sums for undirected Hamiltonicity. In: FOCS, pp. 173–182 (2010)
4. Björklund, A., Husfeldt, T., Taslaman, N.: Shortest cycle through specified elements. In: Rabani, Y. (ed.) SODA, pp. 1747–1753. SIAM (2012)
5. Bodlaender, H.L., Cygan, M., Kratsch, S., Nederlof, J.: Solving weighted and counting variants of connectivity problems parameterized by treewidth deterministically in single exponential time. CoRR, abs/1211.1505 (2012)
6. Cao, Y., Chen, J., Liu, Y.: On feedback vertex set new measure and new structures. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 93–104. Springer, Heidelberg (2010)
7. Cygan, M., Kratsch, S., Nederlof, J.: Fast Hamiltonicity checking via bases of perfect matchings (2012) (to appear at STOC 2013)
8. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: FOCS, pp. 150–159 (2011)
9. Dorn, F., Fomin, F.V., Thilikos, D.M.: Catalan structures and dynamic programming in H-minor-free graphs. J. Comput. Syst. Sci. 78(5), 1606–1622 (2012)
10. Eppstein, D.: The traveling salesman problem for cubic graphs. J. Graph Algorithms Appl. 11(1), 61–81 (2007)
11. Göös, M., Suomela, J.: Locally checkable proofs. In: PODC, pp. 159–168 (2011)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. (2001)
13. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. 63(4), 512–530 (2001)
14. Kleinberg, J., Tardos, É.: Algorithm Design (2005)
15. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 575–586. Springer, Heidelberg (2008)
16. Koutis, I., Williams, R.: Limits and applications of group algebras for parameterized problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 653–664. Springer, Heidelberg (2009)
17. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: SODA, pp. 777–789 (2011)
18. Lovász, L.: On determinants, matchings, and random algorithms. In: FCT, pp. 565–574 (1979)
19. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. Combinatorica 7(1), 105–113 (1987)
20. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms (2002)
21. Raz, R., Spieker, B.: On the "log rank"-conjecture in communication complexity. Combinatorica 15(4), 567–588 (1995)
22. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. J. Comb. Theory 36(1), 49–64 (1984)
23. Williams, R.: Finding paths of length k in $\mathcal{O}^\star(2^k)$ time. Inf. Process. Lett. 109(6), 315–318 (2009)
24. Williams, V.V.: Multiplying matrices faster than coppersmith-winograd. In: STOC, pp. 887–898 (2012)